

# Quantum computation

Samuel L. Braunstein

*Computer Science, University of York, York YO10 5DD, UK*

- Introduction
- 1. Computing at the atomic scale
- 2. Reversible computation
- 3. Classical universal machines and logic gates
- 3.1. FANOUT and ERASE
- 3.2. Computation without ERASE
- 4. Elementary quantum notation
- 5. Logic gates for quantum bits
- 6. Logic gates in the laboratory
- 7. Model quantum computer and quantum code
- 8. Quantum parallelism: Period of a sequence
- 9. The complexity of factoring
- 10. Security and RSA
- 11. Shor's result: Factoring numbers
- 12. Quantum error correction
- 13. Prospects
- 14. Glossary
- Appendix
- Works cited

## INTRODUCTION

A quantum computer is a device that can arbitrarily manipulate the quantum state of a part of itself. The field of quantum computation is largely a body of theoretical promises for some impressively fast algorithms which could be executed on quantum computers. However, since the first significant algorithm was proposed in 1994 [1] experimental progress has been rapid with several schemes yielding two [2, 3] and three quantum-bit manipulations [4]. At the writing of this article it does not seem unreasonable to expect that small quantum computers capable of manipulating the quantum states of five or six two-level systems will be available within around five years. In addition, with the discovery of quantum error correction schemes [5] such machines have the promise of providing long term storage of quantum information and possibly allowing the ability to manipulate many more bits. It is still too early to tell whether the promises of rapid computation are achievable, nor is it yet well understood how broad a class of problems could be significantly speeded up by quantum computers.

Quantum computers were first discussed by Benioff [6] in the context of simulating classical Turing machines (very elementary conventional computers) with quantum unitary evolution. Feynman [7] considered the converse

question of how well classical computers can simulate quantum systems. He concluded that classical computers invariably suffer from an exponential slow-down in trying to simulate quantum systems, but that quantum systems could, in principle, simulate each other without this slow-down. It was Deutsch [8], however, who first suggested that quantum superposition might allow quantum evolution to perform many classical computations in parallel.

To demonstrate where such capabilities may lie hidden, we review an elementary quantum mechanical experiment [9]. The two-slit experiment is prototypic for observing one key feature of quantum mechanics: A source emits photons, electrons or other particles that arrive at a pair of slits. These particles undergo unitary evolution and finally measurement. We see an interference pattern, with both slits open, which wholly vanishes if either slit is covered. In some sense, each particle passes through both slits in parallel. If such unitary evolution were to represent a calculation (or an operation within a calculation) then the quantum system would be performing computations in parallel. In some sense this quantum parallelism comes for free without our having to construct many copies of the 'processing unit.' The output of this system would be given by the constructive interference among the parallel computations.

In this paper we give a tutorial on how quantum mechanics can be used to improve computation. We shall concentrate on the only known algorithm which demonstrates an exponential speedup relative to the best known classical algorithms. Our challenge: solving an exponentially difficult problem for a conventional computer—that of factoring a large number. As a prelude, we review the standard tools of computation, universal gates and machines. These ideas are then applied first to classical, dissipationless computers and then to quantum computers. A schematic model of a quantum computer is described as well as some of the subtleties in its programming. The Shor algorithm [1, 10] for efficiently factoring numbers on a quantum computer is presented in two parts: the quantum procedure within the algorithm and the classical algorithm that calls the quantum procedure. The mathematical structure within the factoring problem is discussed, making it clear what contribution the quantum computer makes to the calculation. The complexity of the Shor algorithm is compared to that of factoring on conventional machines and its relevance to public key cryptography is noted. In addition, we discuss the experimental status of the field and also quantum error correction which may in the long run help solve some the

most pressing difficulties. We conclude with an outlook to the feasibility and prospects for quantum computation in the coming years.

## 1. COMPUTING AT THE ATOMIC SCALE

Quantum computers will perform computations at the atomic scale [9, 11]. We might ask at this point how close conventional computations are to this scale already? Fig. 1 shows a survey made by Keyes in 1988 [12]: The number of dopant impurities in the bases of bipolar transistors used for digital logic against the year. This plot may be thought of as showing the number of electrons required to store a single bit of information. An extrapolation of the plot suggests that we might be within reach of the atomic-scale computations within the next two decades.

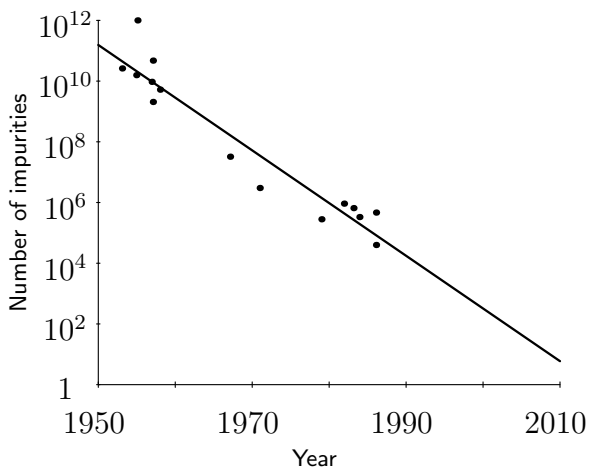


FIG. 1: Plot from Ref. [12] showing the number of dopant impurities involved in logic in bipolar transistors with year. (Copyright 1988 by International Business Machines Corporation, reprinted with permission.)

Another way of viewing this plot is perhaps even more relevant for the development of quantum computation: Conventional computers have been improving in speed and miniaturization at an exponential rate since their earliest days. Clearly there is a bound to our ability to miniaturize conventional electronics and we will likely be touching that limit within the next twenty years. The question is raised, can we continue to expect to see an exponential improvement in performance twenty and more years from now? As we approach some of the physical limits to conventional computational construction we may begin to see a slow-down of this exponential rate. A detailed study of quantum computation may help us understand the fundamental physical limitations upon computation, conventional or otherwise.

## 2. REVERSIBLE COMPUTATION

What are the difficulties in trying to build a classical computing machine on such a small scale? One of the biggest problems with the program of miniaturizing conventional computers is the difficulty of dissipated heat. As early as 1961 Landauer studied the physical limitations placed on computation from dissipation [13]. Surprisingly, he was able to show that all but one operation required in computation could be performed in a reversible manner, thus dissipating no heat! The first condition for any deterministic device to be reversible is that its input and output be uniquely retrievable from each other. This is called logical reversibility. If, in addition to being logically reversible, a device can actually run backwards then it is called physically reversible and the second law of thermodynamics guarantees that it dissipates no heat.

The work on classical, reversible computation laid the foundation for the development of quantum mechanical computers. On a quantum computer, programs are executed by unitary evolution of an input that is given by the state of the system. Since all unitary operators  $U$  are invertible with  $U^{-1} = U^\dagger$ , we can always “uncompute” (reverse) a computation on a quantum computer.

## 3. CLASSICAL UNIVERSAL MACHINES AND LOGIC GATES

We now review the basic logic elements used in computation and explain how conventional computers may be used for any “reasonable” computation. A reasonable computation is one that may be written in terms of some (possibly large) Boolean expression, and any Boolean expression may be constructed out of a fixed set of logic gates. Such a set (e.g., AND, OR and NOT) is called universal. In fact we can get by with only two gates, such as AND and NOT or OR and NOT. Alternatively, we may replace some of these primitive gates by others, such as the exclusive-OR (called XOR or often controlled-NOT); then AND and XOR form a universal set. The truth tables for these gates are displayed in Table I. Any machine which can build up arbitrary combinations of logic gates from a universal set is then a universal computer. Further, which universal set of gates is chosen makes little difference: A theorem by Muller states that the complexity of the simplest circuits needed to compute any reasonable Boolean function is affected by at most a constant multiplicative factor [14].

Which of the above gates is reversible? Since AND, OR, and XOR are many-to-one operations information is lost and they are not, as they stand, logically reversible. Before we discuss how these logic gates may be made reversible we consider some non-standard gates that we shall require.

$A$	$B$	AND	OR	XOR	NOT $B$
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	1
1	1	1	1	0	0

TABLE I: Truth table defining the operation of some simple logic gates. Each row shows two input values  $A$  and  $B$  and the corresponding output values for gates AND, OR and XOR. The output for the NOT gate is shown only for input  $B$ .

### 3.1. FANOUT and ERASE

Although the above gates are sufficient for the mathematics of logic, they are not sufficient to build a machine. A useful computer will also require the FANOUT and ERASE gates (Fig. 2).

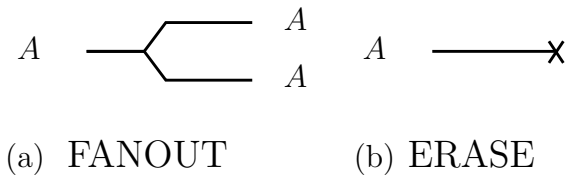


FIG. 2: Two non-standard gates that are required to build a computer, in addition to a universal set of logic gates, are: (a) the FANOUT gate which duplicates an input  $A$  and (b) the ERASE gate which deletes its input.

First consider the FANOUT gate: Is it reversible? Certainly no information has been destroyed so it is at least logically reversible. Landauer showed that it could also be physically reversible [13]. Let us describe a simple model for FANOUT based on Bennett’s scheme for a reversible measurement (Fig. 3) [15] (We note, however, that the concepts involved in this scheme come from Fredkin and Toffoli [16, 17].) Here a dark ball is used to determine the presence or absence of a second (light) ball inside a trap. The trap consists of a set of mirrors and may be thought of as a one-bit memory register. If the trap is occupied then the dark ball is reflected and leaves along direction  $M$  (with the light ball continuing along its original trajectory); otherwise it passes unhindered towards  $N$ . Upon leaving the trap, the dark ball’s direction is used to populate, or not, another trap.

Let us now consider the ERASE operation, required to “clean out” the computer’s memory periodically. One type of erasure can be performed reversibly: If we have a backup copy of some information, we can erase further copies by uncomputing the FANOUT gate. The difficulty arises when we wish to erase our last copy, referred to here as the primitive ERASE.

Consider a single bit represented by a pair of equally probable classical states of some particle. To erase the information about the particle’s state we must irreversibly compress phase-space by a factor of two. If we allowed this compressed phase-space to adiabatically expand, at

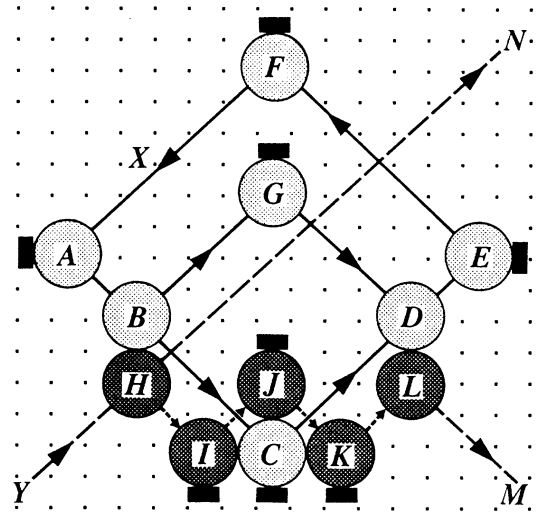


FIG. 3: A reversible measurement of the existence of a (light) ball in a trap of mirrors (dark rectangles) [15]. A (dark) ball enters the trap from  $Y$ . In the absence of a light ball in the trap the dark ball will follow the path  $HN$ . In presence of a light ball (timed to start at  $X$ ) the dark ball will deflect the light one from its unhindered trajectory  $ABCDEF$  to  $ABGDEF$  and will follow the path  $HIJKLM$  itself. (Copyright 1988 by International Business Machines Corporation, reprinted with permission.)

temperature  $T$ , to its original size, we could obtain an amount of work equal to  $k_B T \ln 2$  (where  $k_B$  is Boltzmann’s constant). Landauer concluded, based on simple models and more general arguments about the compression of phase-space, that the erasure of a bit of information at temperature  $T$  requires the dissipation of at least  $k_B T \ln 2$  heat (a result known as Landauer’s principle) [13].

### 3.2. Computation without ERASE

Fortunately, the primitive ERASE is not absolutely essential in computation. To see why, consider what is required to compute arbitrary functions using reversible logic (where the primitive ERASE is forbidden). Landauer showed how any function  $f(a)$  could be made one-to-one by keeping a copy of the input:

$$f : a \rightarrow (a, f(a)) .$$

Here the bold parentheses represent an ordered set of values, in this case, two. Extra “slots” will be added (or removed) as required in our discussion below.

How can this trick be used to perform reversible logic? One solution, developed by Toffoli and Fredkin and known as the Toffoli gate, is shown in Fig. 4 [13, 16, 17]. The output of this gate may be decomposed into various

gates:

$$B \oplus (A.C) = \begin{cases} A.C, & \text{for } B = 0 & \text{(AND)} \\ A \oplus B, & \text{for } C = 1 & \text{(XOR)} \\ \bar{A}, & \text{for } B = C = 1 & \text{(NOT)} \\ A, & \text{for } B \neq C = 1 & \text{(FANOUT)} \end{cases}$$

where  $A.B$  represents an AND gate,  $A \oplus B$  represents an XOR gate and  $\bar{A}$  represents a NOT gate. We see that this gate is universal, because it performs AND, XOR, NOT or FANOUT depending on its inputs. A combination of many such gates could then be used for any computation and would still be reversible.

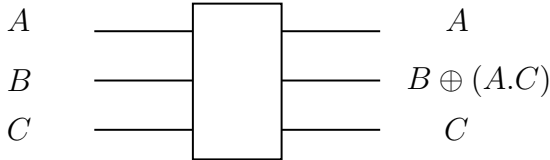


FIG. 4: Three-input three-output universal reversible Toffoli gate. This gate is clearly reversible since a second application of it retrieves the original input.

As noted by Landauer, this procedure leads to an immediate problem because of the absence of the primitive ERASE. The more gates we employ, the more “junk” bits we accumulate: At each gate we must save input bits in order to preserve reversibility. In other words a computer built out of reversible logic instead of conventional, irreversible logic gates would behave like

$$f : a \rightarrow (a, j(a), f(a)) ,$$

with many extra junk bits  $j(a)$ .

Bennett solved this problem by showing that the junk bits could be *reversibly* erased at intermediate steps with minimal run-time and memory costs [18–20]. The spirit of Bennett’s solution may be understood in terms of the following procedure:

$$\begin{aligned} f : a & \rightarrow (a, j(a), f(a)) \\ \text{FANOUT} : (a, j(a), f(a)) & \rightarrow (a, j(a), f(a), f(a)) \\ f^\dagger : (a, j(a), f(a), f(a)) & \rightarrow (a, f(a)) , \end{aligned}$$

where  $f^\dagger$  denotes uncomputing  $f$ , as opposed to computing  $f^{-1}$ . First,  $f$  is computed, producing both junk bits and the desired output. Then the FANOUT gate is applied to duplicate the output. Finally, we uncompute the original function  $f$  by running its computation backwards. This procedure removes the junk bits and the original output. The duplicate, however, remains!

This completes our discussion of the construction of classical, reversible computers. We have found that reversibility does not bar the logical design of computing machines. Before mapping these ideas to quantum systems, however, we introduce some elementary quantum mechanical notation.

#### 4. ELEMENTARY QUANTUM NOTATION

A simple quantum system is the two-level spin- $\frac{1}{2}$  particle. Its basis states, spin-down  $|\downarrow\rangle$  and spin-up  $|\uparrow\rangle$ , may be relabelled to represent binary zero and one, i.e.,  $|0\rangle$  and  $|1\rangle$ , respectively. The state of a single such particle is described by the wavefunction  $\psi = \alpha|0\rangle + \beta|1\rangle$ , i.e., a linear *superposition* amongst any of the possible ‘classical’ states of the system. The squares of the complex coefficients  $|\alpha|^2$  and  $|\beta|^2$  represent the probabilities for finding the particle in the corresponding states. Generalizing this to a set of  $k$  spin- $\frac{1}{2}$  particles we find that there are now  $2^k$  basis states (quantum mechanical vectors that span a Hilbert space) corresponding say to the  $2^k$  possible bit-strings of length  $k$ . Freely moving between decimal, binary and spin labels then we might write, for example for  $k = 5$ , a state  $|25\rangle = |11001\rangle = |\uparrow\uparrow\downarrow\downarrow\uparrow\rangle$ .

The dimensionality of the Hilbert space grows exponentially with  $k$ . In some very real sense quantum computations make use of this enormous size latent in even the smallest systems.

#### 5. LOGIC GATES FOR QUANTUM BITS

In this section we describe how arbitrary logic gates may be constructed for quantum bits. We start by considering various one-bit unitary operations and a single two-bit one—the XOR operation. Combinations of these are sufficient to construct a Toffoli gate for quantum bits or indeed *any* unitary operation on a finite number of bits.

Start with a single quantum bit. If we represent the states  $|\downarrow\rangle$  and  $|\uparrow\rangle$  (i.e.,  $|0\rangle$  and  $|1\rangle$ ) as the vectors  $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$  and  $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ , respectively, then the most general unitary transformation corresponds to a  $2 \times 2$  matrix of the form

$$U_\theta \equiv \begin{pmatrix} e^{i(\delta+\sigma/2+\tau/2)} \cos(\theta/2) & e^{i(\delta+\sigma/2-\tau/2)} \sin(\theta/2) \\ -e^{i(\delta-\sigma/2+\tau/2)} \sin(\theta/2) & e^{i(\delta-\sigma/2-\tau/2)} \cos(\theta/2) \end{pmatrix} ,$$

where we typically take  $\delta = \sigma = \tau = 0$  [21]. Using this operator we can flip bits via:

$$U_\pi|0\rangle = -|1\rangle , \text{ and } U_\pi|1\rangle = |0\rangle .$$

The extraneous sign represents a phase factor that does not affect the logical operation of the gates and may be removed if we wish, now or at a later stage. Such one-bit computations are illustrated schematically as a quantum circuit in Fig. 5 [21, 22].

Another important one-bit gate is  $U_{-\pi/2}$  which maps a spin-down particle

$$U_{-\pi/2}|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) ,$$

to an equal superposition of down and up. Consider a string of  $k$  spin- $\frac{1}{2}$  particles initially spin-down. If we

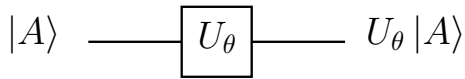


FIG. 5: Schematic of the quantum circuit diagram for a one-bit gate. The line represents a single quantum bit (such as a spin- $\frac{1}{2}$  particle). Initially, this bit has a state described by  $|A\rangle$ ; after it has “passed” through this circuit it comes out in the state  $U_\theta |A\rangle$ .

apply this gate independently to each particle we obtain a superposition of every possible bit-string of length  $k$ :

$$|0\rangle \longrightarrow \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\rangle,$$

where  $q = 2^k$ . Our computer is now in a superposition of an exponentially large number of integers  $a$  from 0 to  $2^k - 1$ . Suppose further, that we could construct a unitary operation which maps a pair of bit-strings  $|a; 0\rangle$  into the pair  $|a; f(a)\rangle$  for some function  $f(a)$ . Then such a unitary operator acting on the superposition of states

$$\frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a; 0\rangle \longrightarrow \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a; f(a)\rangle,$$

would compute  $f(a)$  in parallel an exponentially large number of times for the various inputs  $a$ . This follows from the linearity of quantum mechanics.

To see how such unitary operators may be constructed from a few elementary ones we must also consider the XOR gate [21, 22]. Writing the two-particle basis states as the vectors

$$\begin{aligned} |00\rangle &= \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, & |01\rangle &= \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \\ |10\rangle &= \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, & |11\rangle &= \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \end{aligned}$$

we may represent the XOR gate as a unitary operator

$$U_{\text{XOR}} \equiv \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

Here the first particle acts as a conditional gate to flip the state of the second particle. It is easy to check that the state of the second particle corresponds to the action of the XOR gate given in Table I. The quantum circuit for an XOR gate is illustrated in Fig. 6. This circuit is equivalent to the elementary instruction

$$\text{if } (|A\rangle = 1) |B\rangle = \text{NOT } |B\rangle$$

which may be thought of as example of quantum computer code [23]. The ket-brackets  $| \rangle$  are reminders that we are dealing with quantum rather than classical bits. The XOR gate allows us to move information around as is illustrated in Fig. 7.

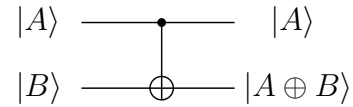


FIG. 6: Quantum circuit diagram for an XOR gate. The lower bit  $|B\rangle$  is flipped whenever the upper bit  $|A\rangle$  is set.

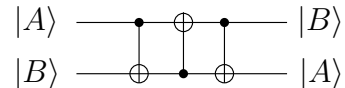


FIG. 7: Circuit for swapping a pair of bits.

How do we construct the Toffoli gate? One major problem with this gate is that it requires three bits in and three out. Quantum mechanically, this seems to correspond to a scattering process involving three-particle collisions [24] calling for a (possibly) unreasonable control of the particles [9]. Fortunately, the Toffoli gate may be constructed by two-particle scattering processes alone [22, 25–28]. In particular, we show a construction here involving the XOR gate and some one-bit gates  $U_\theta$  (Fig. 8) [21]. Not only is the XOR sufficient for all logic operations on a quantum computer, but it can be used to construct arbitrary unitary transformations on any finite set of bits. Numerous proposals for producing such gates have been considered [10, 11] and we discuss some promising experimental results in the next section.

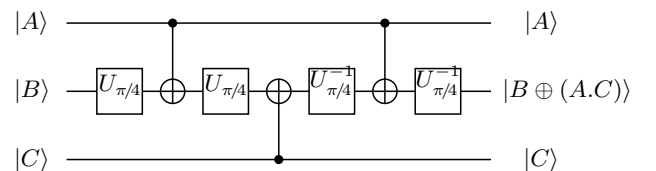


FIG. 8: Toffoli gate built from two-bit XOR gates plus some one-bit gates [9, 21]. This circuit introduces some extra signs in the unitary matrix  $U_{\text{XOR}}$  which may be removed at a later stage.

## 6. LOGIC GATES IN THE LABORATORY

In this section we briefly review two recent experiments which demonstrate conditional dynamics of a type which is promising for constructing quantum logic gates [2, 3]. These two results appeared back-to-back in the same issue of *Physical Review Letters*.

The first experiment, by Turchette *et al* [2], demonstrated that the phase of a weak coherent optical field could be controlled by the intensity of a second coherent field at a slightly different frequency. The chief result is that such a high non-linear susceptibility was achieved that a large phase shift (up to  $16^\circ$ ) was produced by a change in intensity corresponding to a single photon in the second field. The coupling between the optical fields was obtained using the hyperfine level structure of cesium. A stream of Cs atoms was dropped through an optical cavity which effectively restrained the atomic decay modes to the cavity modes. This allowed the atoms to strongly couple to the optical fields passing through the cavity with minimal incoherent emission into free space. Since coherent instead of single-photon states were used in this experiment, however, there could be no direct demonstration of the coherence retained in the final state of the optical fields. In this scheme a qubit would need to be represented by single photon states rather than weak coherent states, but that does not appear to be a great difficulty.

The second experiment by Monroe *et al* [3] involved a direct demonstration of an XOR gate in a radio-frequency ion trap (also known as a Paul trap). Here the lowest vibrational excitation of a single  $^9\text{Be}^+$  ion in the trap and its hyperfine state represented the two qubits. A pair of off-resonant laser beams were used to drive stimulated Raman transitions between the basis states of these two qubits. The XOR gate was executed using three laser pulses as had been suggested by Cirac and Zoller [29]. The coherence of the qubits in this system was reported to have survived for around to 10 to 20 XOR operations [30].

This single-ion trap experiment was based on a theoretical proposal for a complete quantum computer by Cirac and Zoller [31]. They suggested using a linear ion trap to hold a set of ions in a well localized manner by mutual electrostatic repulsion. Each ion would be ‘addressed’ separately their own lasers. By tuning the lasers shining on individual ions to the appropriate levels either single-qubit operations could be performed or the internal ionic state could be transferred to that of the lowest two vibrational modes of the trap. In this way two-qubit gates could be simulated between even non-contiguous ions via their interactions with the traps vibrational modes. This ability significantly reduces the complexity of elementary operations over other proposals [32]. The fact that this theoretical proposal was implemented within a few months, though in a slightly modified form, suggests that few qubit processors could become a reality within a relatively short period of time. In section 13 below we will discuss the short term prospects for such machines.

In quantum computation we normally aim at having the qubits maximally couple to each other and minimally to the outside world. This lessens the actions of environment-induced decoherence. From this perspective the latter scheme involving ion-traps appears more ideal.

However, there is another class of quantum logic processor which aims at communicating quantum information between various, possibly distantly, separated subsystems. This might allow for the combining of smaller quantum computers into larger ones operating on more qubits through combination. It is likely also to be important for the area of quantum communication and potential technologies which few-bit quantum logic processors might enable. For such tasks the former scheme involving ‘flying’ qubits appears a more likely direction. It is possible that a mature quantum computation technology would have components incorporating the positive aspects of both the above schemes.

## 7. MODEL QUANTUM COMPUTER AND QUANTUM CODE

In this section we describe a simple abstract model for a quantum computer based on a classical computer instructing a machine to manipulate a set of spins. This model has some intrinsic limitations which make designing algorithms in a high-level language somewhat tricky. We discuss some of the rules for writing such quantum computer code as a high-level language and give an example.

Consider the following model for the operation of a quantum computer: Several thousand spin- $\frac{1}{2}$  particles (or two-level systems) are initially in some well defined state, such as spin-down. A classical machine takes single spins or pairs of spins and entangles them (performing an elementary one-bit operation  $U_\theta$  or the two-bit XOR gate); see Fig. 9a, b and c. These stages are repeated on different pairs of spins according to the instructions of a conventional computer program. Since the spins are entangled, we must not look at the spins at intermediate stages: We must keep the quantum superposition intact. Furthermore, nothing else may interfere with the spins which could destroy their orientation or interrupt their unitary evolution. Once this well-defined cycle of manipulation is complete the orientations of the spins are measured (Fig. 9d). This set of measured orientations is the output of the computation.

Given this paradigm for a quantum computer, what might its high-level language (its computer code) look like? The most serious difficulty that must be dealt with is that the quantum information is manipulated by a conventional computer in a completely blind manner—without any access to the values of this quantum information. This means that the program cannot utilize “short-cuts” conditional on the value of a quantum variable (or register or bit). For example, loops must be iterated through exactly the same number of times independent of the values of the quantum variables. Similarly, conditional branches around large pieces of code must be broken down into repeated conditions for each step. In addition, each instruction performed upon the quantum bits must be logically reversible. Thus, ordinary assign-

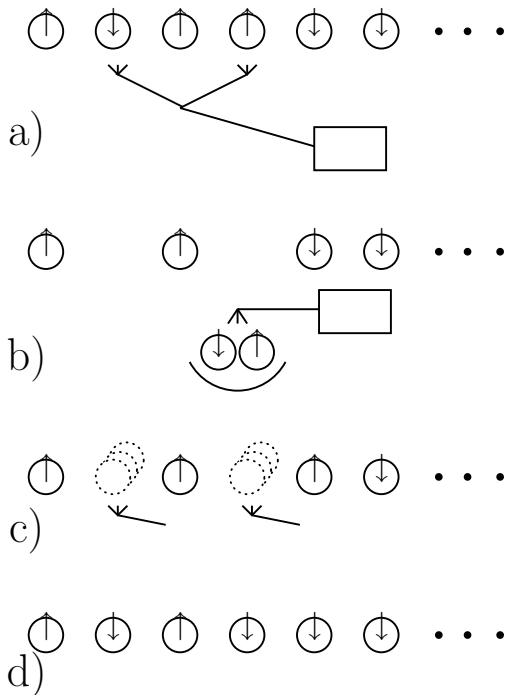


FIG. 9: Model quantum computer as pictured by Shor [33]. Initially all particles are spin-down. Stage a) a classical machine takes a single or pair of spins and in stage b) it performs a selected one-bit or two-bit operation; in stage c) the “entangled” particles are returned to their original locations. These three stages are repeated many times in accord with the instructions given by an ordinary classical computer. When this cycle is complete stage d) consists of measuring the state of the particles (leaving them in some particular bit-string); this bit-string is the result of the computation.

ments of a value to a variable, such as  $|a\rangle = n$ , are not legal and must instead be performed as increments on an initially zeroed variable, such as  $|a\rangle = |a\rangle + n$ .

An example of such code that could run on this machine might look like this [23]:

```

do 10 k = 1, worstdiv
  |a⟩ = |a⟩ - n
  if ( |a⟩ >= 0 ) |q⟩ = |q⟩ + 1
10 continue
do 20 k = 1, worstdiv
  if ( k > |q⟩ ) |a⟩ = |a⟩ + n
20 continue

```

This code fragment could be used to calculate the quotient and the remainder, placed in  $|q\rangle$  and  $|a\rangle$ , respectively, for the division of  $|a\rangle$  by  $n$ ; the constant `worstdiv` is the worst-case number of times the loop must be traversed. Here  $|q\rangle$  is initially zero. Each instruction here is either a conventional computer instruction or one involving some quantum variables. The former are direct instructions for the external computer, while the latter must be interpreted as a sequence of manipulations to be performed upon the quantum bits. As it stands, this

code is *not* reversible (neither is it very efficient), e.g., the label 10 gives no specification of which routes might be used to get to it. It can, however, be easily rewritten [23].

## 8. QUANTUM PARALLELISM: PERIOD OF A SEQUENCE

We now have sufficient ingredients to understand how a quantum computer can perform logical operations and compute just like an ordinary computer. In this section we describe an algorithm which makes use of the quantum parallelism that we have hinted at already: finding the period of a long sequence.

Consider the sequence

$$f(0), f(1), \dots, f(q-1),$$

where  $q \equiv 2^k$ ; we shall use quantum parallelism to find its period. We start with a set of initially spin-down particles which we group into two sets (two quantum registers, or quantum variables):

$$|0; 0\rangle = |\downarrow, \downarrow, \dots; \downarrow, \downarrow, \dots\rangle,$$

the first set having  $k$  bits; the next having sufficient for our needs. (In fact other registers are required, but by applying Bennett’s solution to space management they may be suppressed in our discussion here.) On each bit of the first register we perform the  $U_{-\pi/2}$  one-bit operation, yielding a superposition of every possible bit-string of length  $k$  in this register:

$$\longrightarrow \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a; 0\rangle.$$

The next stage is to break down the computation, corresponding to the function  $f(a)$ , into a set of one-bit and two-bit unitary operations. The sequence of operations is designed to map the state  $|a; 0\rangle$  to the state  $|a; f(a)\rangle$  for any input  $a$ . Now we see that the number of bits required for this second register must be at least sufficient to store the longest result  $f(a)$  for any of these computations. When, however, this sequence of operations is applied to our exponentially large superposition, instead of the single input, we obtain

$$\longrightarrow \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a; f(a)\rangle.$$

An exponentially large amount of computation has been performed essentially for free.

The final computational step, like the first, is again a purely quantum mechanical one. Consider a discrete “quantum” Fourier transform on the first register

$$|a\rangle \longrightarrow \frac{1}{\sqrt{q}} \sum_{c=0}^{q-1} e^{2\pi i ac/q} |c\rangle.$$

It is easy to see that this is reversible via the inverse transform and indeed it is readily verified to be unitary. Further, an efficient way to compute this transform with one-bit and two-bit gates has been described by Coppersmith (Fig. 10) [11, 34, 35].

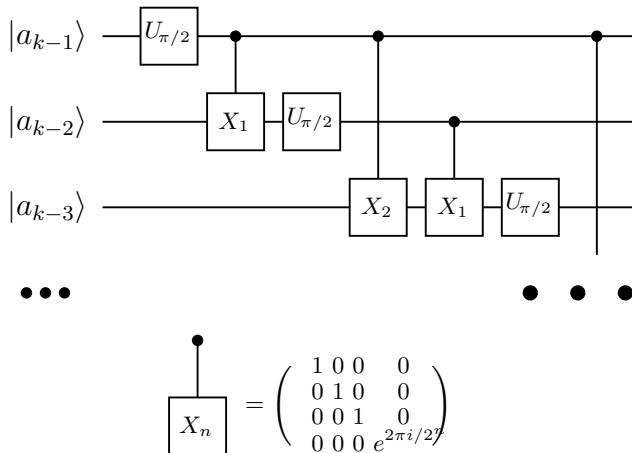


FIG. 10: Circuit for the quantum Fourier transform of the variable  $|a_{k-1}\dots a_1 a_0\rangle$  using Coppersmith’s fast Fourier transformation approach [11, 34, 35]. The two-bit “ $X_n$ ” gate may itself be decomposed into various one-bit and XOR gates [21].

When this quantum Fourier transform is applied to our superposition, we obtain

$$\longrightarrow \frac{1}{q} \sum_{a=0}^{q-1} \sum_{c=0}^{q-1} e^{2\pi i ac/q} |c; f(a)\rangle.$$

The computation is now complete and we retrieve the output from the quantum computer by measuring the state of all spins in the first register (the first  $k$  bits). Indeed, once the Fourier transform has been performed the second register may even be discarded [36].

What will the output look like? Suppose  $f(a)$  has period  $r$  so  $f(a+r) = f(a)$ . The sum over  $a$  will yield constructive interference from the coefficients  $e^{2\pi i ac/q}$  only when  $c/q$  is a multiple of the reciprocal period  $1/r$  [37]. All other values of  $c/q$  will produce destructive interference to a greater or lesser extent. Thus, the probability distribution for finding the first register with various values is shown schematically by Fig. 11.

One complete run of the quantum computer yields a random value of  $c/q$  underneath one of the peaks in the probability of each result  $\text{prob}(c)$ . That is, we obtain a random multiple of the inverse period. To extract the period itself we need only repeat this quantum computation roughly  $\log \log r/k$  times in order to have a high probability for at least one of the multiples to be relatively prime to the period  $r$ —uniquely determining it [1]. Thus, this algorithm yields only a probabilistic result. Fortunately, we can make this probability as high as we like.

All the above work may appear a little anti-climactic. We have gone to a lot of trouble to design a quantum

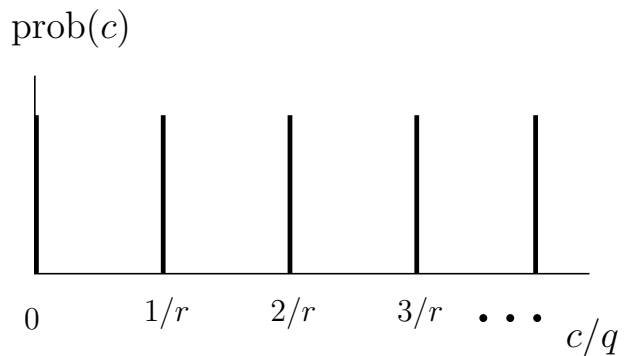


FIG. 11: Idealized plot of the probability of each result  $\text{prob}(c)$  versus  $c/q$ . Constructive interference produces narrow peaks at multiples of the inverse period of the sequence  $1/r$ . (The discrete approximation means that the peaks will actually peaks have a non-zero width [37].)

computer to find the period of a sequence. The point is, however, that the sequence is calculated in parallel and is exponentially long—even for a small value of say  $k = 270$  bits in the first register the quantum computer has calculated and stored more results than there are particles in the universe.

This algorithm for finding the period of an exponentially long sequence on a quantum computer lies at the heart of efficiently factoring numbers. We first proceed to review the computational difficulty of factoring for conventional computers. Then we shall discuss the implications this computational difficulty has had for the secure transmission of private information via public key cryptosystems. We will then follow these discussions with Shor’s new result for efficient factoring [1].

## 9. THE COMPLEXITY OF FACTORING

How can we quantify the difficulty of solving a problem with a conventional computer? Surely once the computer program is written and debugged we may simply let it run and wait for the answer. But this brings us to the crux of the difficulty: For a given problem how long must we expect to wait for the solution? When more carefully phrased this becomes the simplest measure of computational difficulty of an algorithm, yielding the ‘algorithmic complexity’ of the problem.

To be more specific, without getting into technicalities, let us consider the problem of factoring a number  $N$  into its prime factors (e.g., the number 51688 may be decomposed as  $2^3 \times 7 \times 13 \times 71$ ). A convenient way to quantify how quickly a particular algorithm may solve this, or any, problem is to ask how the number of steps to complete the algorithm scales with the size of the “input” the algorithm is fed. For the factoring problem, this input is just the number  $N$  we wish to factor; hence the length of the input is  $\log N$ . (The base of the logarithm is determined



by our numbering system. Thus a base of 2 gives the length in binary; a base of 10 in decimal. For example, the number 51688 requires 16 binary digits, but only five decimal digits to specify it.) ‘Reasonable’ algorithms are ones which scale as some small-degree polynomial in the input size (with a degree of perhaps 2 or 3). One famous example of a fast algorithm is the Fast Fourier Transform which requires roughly  $O(M \log_2 M)$  steps to perform the discrete Fourier transform of  $M$  points (so for a fixed precision the input scales as  $M$ ); by contrast, a conceptually simpler algorithm equivalent to matrix multiplication would require  $O(M^2)$  computational steps [38]. This modest improvement from a quadratic to a roughly linear complexity has made many image processing applications possible with even quite modest computers.

On conventional computers the best known factoring algorithm runs in  $O(\exp[(64/9)^{1/3}(\ln N)^{1/3}(\ln \ln N)^{2/3}])$  steps [39]. This algorithm, therefore, scales exponentially with the input size  $\log N$ . For instance, in 1994 a 129 digit number (known as RSA129 [40]) was successfully factored using this algorithm on approximately 1600 workstations scattered around the world; the entire factorization took eight months [41]. Using this to estimate the prefactor of the above exponential scaling, we find that it would take roughly 800,000 years to factor a 250 digit number with the same computer power; similarly, a 1000 digit number would require  $10^{25}$  years (significantly longer than the age of the universe). This difficulty is, however, almost certainly exaggerated since it takes no account of the constant improvement in factoring algorithms and the constant speedup of computer hardware. In fact, both of these components have shown a more or less exponential improvement over the last few decades [39], with each contributing roughly equally to the increased computational power. The difficulty of factoring large numbers is crucial for public-key cryptosystems, such as ones used by banks. There, such codes rely on the difficulty of factoring numbers with around 250 digits.

## 10. SECURITY AND RSA

Cryptography as a discipline aims at minimizing the affect of the dishonest. One such situation is the need for secure communication between two parties across an insecure channel. The sender encrypts a *plaintext* message with an encryption key yielding the *cyphertext*. The message is sent across an insecure media where we must assume an eavesdropper may have access. The receiver takes the cyphertext and uses a decryption algorithm to restore the plaintext. If we assume that the eavesdropper has access to the decryption algorithm then the receiver must have something in addition to ensure the security of the transmitted message; this is the decryption key. According to Shannon’s information theory the cyphertext must contain some information about the plaintext message unless the decryption key is at least as long as the

message itself [42]. Such a perfect cipher was invented in 1917, and is known as the Vernam cipher or one-time pad; it requires a key equal in size to the plaintext message. Thus, for perfect security we have the problem of distributing the key itself, which must be done over a secure channel such as by trusted courier. In many situations, such as banking transactions where the volume of information is very large, this is unreasonable.

An understanding of computational complexity allows us to ‘circumvent’ the restriction of Shannon’s theory: A pseudorandom number generator can be used to generate a long almost random key from a much smaller secret key. If the lack of randomness cannot be discovered except through an unreasonable amount of computational effort by the eavesdropper then we have a secure encryption system. An excellent example of such a scheme is the U.S. Data Encryption Standard (DES) which uses a 56 bit key [43]. One estimate of the security of DES suggests that for around a million dollars a special purpose machine could be built to try all  $2^{56}$  keys in a few hours, though security can be easily enhanced by multiple application and a larger effective key [44]. Clearly short keys reduce the burden of key distribution amongst single pairs of users, but for  $n$  users  $n(n-1)/2$  keys would be required to allow any pair to communicate securely. This becomes unwieldy for commercial applications where millions of users may be involved.

Another approach, also based on computational complexity, is known as public key encryption. The most popular scheme, and one used in many commercial applications, is RSA encryption [40]. A person wishing to receive secret communications simply publishes a pair of numbers  $(N, e)$  which form the public key. Encryption involves converting the message to numerical data and dividing it into blocks of numbers  $m_j$  each smaller than  $N$ . Each block  $m_j$  of the message is then encrypted by its modular exponentiation

$$c_j \equiv m_j^e \pmod{N},$$

where  $\text{mod } N$  represents modulo arithmetic (the expression is computed and only the remainder after division by  $N$  is retained). The encrypted blocks  $c_j$  are then transmitted to the receiver via a public channel. Thus, RSA (and any other public key scheme) efficiently solves the key distribution problem.

Decryption by the receiver requires knowing the inverse operation, i.e., knowing the  $d$  such that

$$m_j \equiv c_j^d \pmod{N},$$

reconstructs the original message. The size of  $N$  makes the direct determination of  $d$  too difficult. Instead,  $d$  is constructed along with the public key pair  $(N, e)$  in an efficient manner. This construction involves choosing  $N = pq$  as the product of a pair of comparably sized primes, with  $e$  relatively prime to both  $p-1$  and  $q-1$ , and solving the much simpler problem:

$$\begin{aligned} ed &\equiv 1 \pmod{p-1} \\ ed &\equiv 1 \pmod{q-1}. \end{aligned}$$

It is important to note that the best algorithms for finding  $d$  proceed by first factoring  $N$ , thus the security of RSA relies on the assumed difficulty of factoring [45].

## 11. SHOR'S RESULT: FACTORING NUMBERS

Recently, an algorithm was developed by Peter Shor of AT&T for factoring numbers on a quantum computer which runs in  $O((\log N)^3)$  steps [1, 46]. This is cubic in the input size, so factoring a 250 digit number with such an algorithm would require only a few billion steps. The implication is that public key cryptosystems based on factoring may be breakable. In this section we give the classical portion of Shor's algorithm which relates factoring to finding the period of an exponentially long sequence and hence makes the problem tractable for a quantum computer.

We wish to factor the number  $N$ . It will be sufficient to find even a single factor since then we can reduce the problem to a simpler one. First, select a number  $x$ . Euclid's algorithm (see appendix) could be used to efficiently compute the common factors between  $N$  and  $x$ , hence reducing our problem. We, therefore, assume that these numbers are co-prime. Next, consider the sequence formed by the function  $f(a) = x^a \pmod{N}$ . This sequence has the form:

$$\begin{array}{c} 1, x, \dots, x^{r-1}, x^r, x^{r+1}, \dots \\ \underbrace{1, x, \dots, x^{r-1}}_{r\text{-terms}}, \underbrace{1, x, \dots}_{r\text{-terms}}, \underbrace{1, x, \dots}_{r\text{-terms}} \end{array}$$

Here the top row is just the sequence of powers  $\{x^a\}$ ; the bottom row is the same sequence written in modulo arithmetic, namely  $\{x^a \pmod{N}\}$ . The number  $r$  is just the first non-trivial power where  $x^r \equiv 1 \pmod{N}$ . A close look at this sequence shows that it has a periodic structure with period  $r$ . Using standard algorithms this period would not be readily accessible for a long sequence. However, with the quantum computer algorithm described in section 8 it could be calculated efficiently. This possibility opens up a novel way to find the factors of  $N$  as we shall now describe.

Let us suppose that we have obtained the period  $r$  by the above quantum computer algorithm [47]. If this period is even we may proceed with our factoring algorithm. If not, we must select another  $x$  and start again. A randomly chosen  $x$  will yield a suitably even period  $r$  fifty-percent of the time so not too many trials will be needed [1, 10].

We now proceed with the algorithm: Having chosen an  $x$  so that the sequence  $\{x^a \pmod{N}\}$  has an even period  $r$ , we rewrite the expression  $x^r \equiv 1 \pmod{N}$  as the difference of two squares:

$$(x^{r/2})^2 - 1 \equiv 0 \pmod{N}.$$

Expressing the left-hand-side as a product between a sum and difference we obtain

$$(x^{r/2} + 1)(x^{r/2} - 1) \equiv 0 \pmod{N}.$$

This simply says that the product of the two terms on the left is a multiple of the number  $N$  we wish to factor. Thus, either one or the other of these terms must have a factor in common with  $N$ . The final step in the algorithm then is to calculate the greatest common divisor of these terms individually with  $N$  (see the appendix for an efficient classical algorithm); any non-trivial common divisor will be a factor we have sought, thus completing our search.

As an example, consider the number  $N = 91$ . Choosing  $x = 3$  we find that the sequence  $3^a \pmod{91}$  has the form:

$$\begin{array}{l} a : 0, 1, 2, 3, 4, 5, 6, 7, \dots \\ 3^a : 1, 3, 9, 27, 81, 243, 729, 2187, \dots \\ 3^a \pmod{91} : 1, 3, 9, 27, 81, 61, 1, 3, \dots \end{array}$$

A quantum computer could calculate the period in parallel, however, it is sufficient here to see by eye that this sequence has a period of  $r = 6$  (since it is even we may proceed with the algorithm). Rearranging the expression  $3^6 \equiv 1 \pmod{91}$  as discussed above we conclude that  $28 \times 26 \equiv 0 \pmod{91}$ . This implies that either  $\text{gcd}(28, 91)$  or  $\text{gcd}(26, 91)$  will be a non-trivial factor of 91 (where  $\text{gcd}$  is the greatest common divisor function). In fact, in this case, both terms yield a different factor, 7 and 13, respectively. This completes the prime factorization of 91 yielding:  $91 = 7 \times 13$ .

## 12. QUANTUM ERROR CORRECTION

Building a quantum computer is a daunting task. Even within apparently small atomic-scale systems, quantum computation runs on the enormous size of Hilbert space. Quantum computation involves building a trajectory from a standard initial state to a complex final state. The main difficulty is keeping to this trajectory. To fail is to be lost in Hilbert space. The largest problem is hypersensitivity to perturbations, shifting the computational trajectory randomly from its path. Such perturbations come from an unintentional coupling to external noise [48]. In this section we briefly touch on quantum error correction and fault tolerant computing, both introduced by Shor [5, 49], which promise to greatly alleviate the problem associated with unwanted perturbations.

In straight quantum error correction the state of a fragile quantum system is *encoded* into a quantum system having more degrees of freedom. By choosing the mapping to a suitable subspace of the larger system a limited class of errors which occur on this larger space may be completely removed. In Fig. 12 we see a circuit for Shor's original scheme. [Since then much work has been done in the last year [50–55] (to cite just a few)]. The unprotected single qubit  $|\psi\rangle$  is processed by the left half of the circuit shown in Fig. 12 until just before the shaded region. The resulting combined 9-qubit state represents the now error protected encoded state. Any single qubit error (represented by the shaded region) on this encoded

state may now be ‘undone’ by sending the state through the decoding and correcting circuit shown to the right of the shaded region.

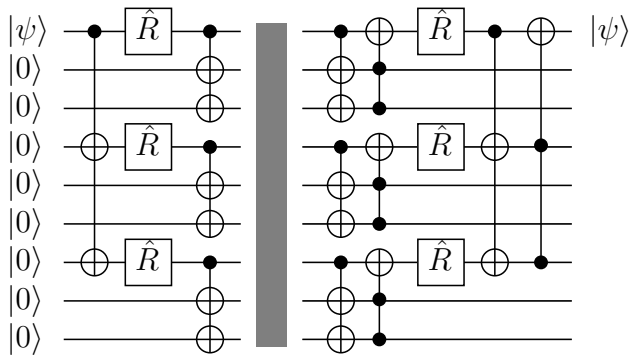


FIG. 12: Schematic of Shor’s original error correction scheme to protect one qubit in a 9-qubit code against arbitrary 1-qubit decoherence [5]. The unprotected qubit enters as qubit from the left. It’s error protection encoded state is just to the left of the shaded region. This region represents an arbitrary 1-qubit error which may be introduced due to coupling to the environment. To recover the original qubit one performs the remainder of the circuit to the right of the decoherence. (The text describes the new circuit elements.)

We have introduced three new circuit elements in Fig. 12 which we now explain: The first 3-qubit gate involves a single control qubit (at the heavy dot) from qubit 1 to qubits 4 and 7. This gate is a shorthand notation for a pair of XOR gates (or controlled-NOT gates) between gates 1 and 4 and gates 1 and 7. The second new 3-qubit gate involves two control qubits (see for example the last gate in the decoding circuit). This is a Toffoli gate with the condition to flip qubit 1 given by the logical AND of qubits 4 and 7 (recall that these logical operations occur separately for each branch of the wavefunction). (In this circuit a Toffoli gate without extraneous phases is required.) Finally, the single qubit gate labelled  $\hat{R}$  is defined by

$$\begin{aligned}\hat{R}|0\rangle &= (|0\rangle + |1\rangle)/\sqrt{2} \\ \hat{R}|1\rangle &= (|0\rangle - |1\rangle)/\sqrt{2},\end{aligned}$$

which is a variation of the operation  $\hat{U}_{-\pi/2}$  discussed earlier.

The above scheme and its variations allow for the long term storage of quantum information. Unfortunately, they require that the encoding and decoding circuitry operate without error. Further, the above circuitry only slows the rate of decoherence, but does not eliminate its effect. By contrast, fault tolerant quantum computation can get around these problems and, in principle, allows for unlimited quantum computation provided a *threshold* in accuracy for each elementary gate’s operation in the presence of decoherence can be surpassed. Fault tolerant computation involves a sophisticated redesign of the error correction and computation circuitry so as to allow

for the imperfect operation of individual gates or for a decoherence event during computation. The future success of quantum computation will almost certainly rely quite heavily on these techniques and upon our ability to determine theoretically, and experimentally surpass, the threshold (see Refs. 56).

### 13. PROSPECTS

We now describe the likely prospects for quantum computation. In this paper we discussed a single algorithm yielding an exponential speed-up over conventional methods: Effectively the calculation of the period of a long sequence. To date this is the only algorithm displaying such a speed-up. This algorithm was applied to a traditional computer-science problem, factoring, only by recognizing a deeper structure within that problem. This requirement appears to be a general one: Quantum parallelism will only yield an exponential speedup in problems whose structure avoids the need to try exponentially many solutions [57–60]. Thus, a brute force approach to some of the hardest computational questions, known as NP-complete problems, will not succeed with the aid of quantum parallelism. Any progress for such problems will require finding a deeper structure within them. Thus, it appears likely that quantum computers will be useful for simulating (or manipulating) small quantum systems [7, 11, 61]. Finally, a class of algorithms for quantum computers for data base searching, estimating the median of a data base etc [62], have been discovered which yield a modest improvement: They take roughly the square-root of the number of computational steps as required on a conventional computer.

How difficult will it be to build a quantum computer? Currently, several implementations are being considered by theoreticians and experimentalists worldwide [2–4, 25, 26, 29, 31, 32, 63, 64]. The most promising scheme to date appears to involve linear ion traps [3, 29, 31]. There are already several theoretical studies investigating the limitations to these systems and for numerous reasons it seems that ion trap quantum computers will be limited to computations involving no more than around 10 to 20 ions [65–68]. It seems likely then that the first generation of quantum computers will not be performing traditional computations but will be used for the manipulation of small amounts of quantum information: a quantum information processor employed possibly for quantum cryptography, quantum teleportation, quantum storage and quantum communication of quantum information. Indeed, these non-traditional tasks will probably lead to new kinds of technology even with the relatively modest quantum computers we will be capable of building within a few years.

Over the next couple of decades we will approach computing at the atomic scale. Heat dissipation will become an ever increasing problem. The lessons from reversible classical computation and quantum computation

may help us overcome this engineering hurdle.

#### 14. GLOSSARY

**Qubit:** Quantum bit of information. The amount of information that may be held by a two-level quantum system.

#### APPENDIX

Here we describe Euclid's algorithm for finding the greatest common divisor (gcd) between a pair of numbers  $n_0 \geq n_1$  [69]. The algorithm proceeds by calculating the sequence of divisions with remainder for these numbers:

$$\begin{aligned} n_0 &= d_1 \times n_1 + n_2 \\ n_1 &= d_2 \times n_2 + n_3 \\ &\vdots \\ n_{m-2} &= d_{m-1} \times n_{m-1} + n_m \\ n_{m-1} &= d_m \times n_m + 0, \end{aligned}$$

where the  $d_m$  are the quotients and  $n_{m-1} \geq n_m$  at each stage. The last non-zero remainder  $n_m$  yields the answer, i.e.,  $\text{gcd}(n_0, n_1) = n_m$ . For example, the sequence

$$\begin{aligned} 91 &= 3 \times 28 + 7 \\ 28 &= 4 \times 7 + 0, \end{aligned}$$

shows that  $\text{gcd}(28, 91) = 7$  in just two steps. The worst case number of steps required to complete Euclid's algorithm is  $O(\log \log n_1)$ .

#### Acknowledgements:

The author appreciate discussions with Charles Bennett, Netta Cohen, David DiVincenzo, Richard Jozsa, Peter Shor, Robert Solovay and Umesh Vazirani. This work funded in part by a Feinberg Fellowship while the author was visiting the Chemical Physics Department at the Weizmann Institute of Science, Rehovot, Israel and in part by a Humboldt Fellowship while visiting the Universität Ulm, Germany.

#### REFERENCES

- [1] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proc. 35th Annual Symposium on the Foundations of Computer Science*, edited by S. Goldwasser (IEEE Computer Society Press, Los Alamitos, California, 1994), pp. 124-134.
- [2] Q. A. Turchette, C. J. Hood, W. Lange, H. Mabuchi and H. J. Kimble, "Measurement of conditional phase shifts for quantum logic," *Phys. Rev. Lett.* **75**, 4710-4713 (1995).
- [3] C. Monroe, D. M. Meekhof, B. E. King, W. M. Itano and D. J. Wineland, "Demonstration of a fundamental quantum logic gate," *Phys. Rev. Lett.* **75**, 4714-4717 (1995).
- [4] N. Gershenfeld and I. Chuang, "Bulk spin-resonance quantum computation," *Science* **275**, 350-356 (1997).
- [5] P. W. Shor, "Scheme for reducing decoherence in quantum computer memory," *Phys. Rev. A* **52**, R2493-R2496 (1995).
- [6] P. Benioff, "The computer as a physical system: a microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines," *J. Stat. Phys.* **22** 563-591 (1980); "Quantum mechanical Hamiltonian models of discrete processes," *J. Math. Phys.* **22**, 495-507 (1981); "Quantum mechanical models of Turing machines that dissipate no energy," *Phys. Rev. Lett.* **48**, 1581-1585 (1982); "Quantum mechanical Hamiltonian models of discrete processes that erase their own histories: application to Turing machines," *Int. J. Theor. Phys.* **21**, 177-201 (1982); "Quantum mechanical Hamiltonian models of Turing machines," *J. Stat. Phys.* **29**, 515-546 (1982).
- [7] R. P. Feynman, "Simulating physics with computers," *Int. J. Theor. Phys.* **21**, 467-488 (1982); "Quantum mechanical computers," *Found. Phys.* **16**, 507-531 (1986) [reprinted from *Opt. News* **11**, 11 (1985)].
- [8] D. Deutsch, "Quantum theory as a universal physical theory," *Int. J. Theor. Phys.* **24**, 1-41 (1985); "Quantum theory, the Church-Turing principle and the universal quantum computer," *Proc. Roy. Soc. Lond. A* **400**, 97-117 (1985).
- [9] D. P. DiVincenzo, presented at *Quantum Computation 1994*, Villa Gualino, Turin, Italy, October 1994, unpublished.
- [10] A more technically detailed description of the Shor algorithm may be found in: A. Ekert and R. Jozsa, "Quantum Computation and Shor's Factoring Algorithm," *Rev. Mod. Phys.* **68**(3), July (1996).
- [11] D. P. DiVincenzo, "Quantum computation," *Science*, **270**, 255-261 (1995).
- [12] R. W. Keyes, "Miniaturization of electronics and its limits," *IBM J. Res. Develop.* **32**, 24-28 (1988).
- [13] The seminal paper in reversible computation: R. Landauer, "Irreversibility and heat generation in the computing process," *IBM J. Res. Develop.* **3**, 183-191 (1961).
- [14] D. E. Muller, "Complexity in electronic switching circuits," *IRE Trans. Electr. Comput.* **5**, 15-19 (1956).
- [15] This paper describes the history of reversible computation: C. H. Bennett, "Notes on the history of reversible computation," *IBM J. Res. Develop.* **32**, 16-23 (1988).
- [16] T. Toffoli, "Reversible computing," in *Automata, Languages and Programming*, Eds. J. W. de Bakker and J. van Leeuwen (Springer-Verlag, New York,

- 1980) pp. 632-644.
- [17] E. Fredkin and T. Toffoli, "Conservative logic," *Int. J. Theor. Phys.* **21**, 219-253 (1982).
- [18] C. H. Bennett, "Logical reversibility of computation," *IBM J. Res. Develop.* **17**, 525-532 (1973).
- [19] C. H. Bennett, "Time/space trade-offs for reversible computation," *SIAM J. Comput.* **18**, 766-776 (1989).
- [20] M. Li, J. Tromp and P. Vitanyi, "Analysis of Reversible Simulation of Irreversible Computation by Pebble Games," *Physica D* **120**, 168-176 (1998).
- [21] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin and H. Weinfurter, "Elementary gates for quantum computation," *Phys. Rev. A* **52**, 3457-3467 (1995).
- [22] D. P. DiVincenzo, "Two-bit gates are universal for quantum computation," *Phys. Rev. A* **51**, 1015-1022 (1995).
- [23] D. P. DiVincenzo, private communication and work presented at *Quantum Computation 1995*, Villa Gualino, Turin, Italy, June 1995, unpublished.
- [24] D. Deutsch, "Quantum computational networks," *Proc. Roy. Soc. Lond. A* **425**, 73-90 (1989).
- [25] A. Barenco, D. Deutsch and A. Ekert, "Conditional quantum dynamics and logic gates," *Phys. Rev. Lett.* **74**, 4083-4086 (1995).
- [26] T. Sleator and H. Weinfurter, "Realizable universal quantum gates," *Phys. Rev. Lett.* **74**, 4087-4090 (1995).
- [27] D. Deutsch, A. Barenco and A. Ekert, "Universality of quantum computation," *Proc. Roy. Soc. Lond. A* **449**, 669-677 (1995).
- [28] S. Lloyd, "Almost any quantum logic gate is universal," *Phys. Rev. Lett.* **75**, 346-349 (1995).
- [29] J. I. Cirac and P. Zoller, "Quantum computations with cold trapped ions," *Phys. Rev. Lett.* **74**, 4091-4094 (1995).
- [30] T. Thompson, "When silicon hits its limits, what's next?" *Byte* **21**(4), 45-54 (1996).
- [31] T. Pellizzari, S. A. Gardiner, J. I. Cirac and P. Zoller, "Decoherence, continuous observation and quantum computing: a cavity QED model," *Phys. Rev. Lett.* **75**, 3788-3791 (1995).
- [32] S. Lloyd, "A potentially realizable quantum computer," *Science* **261**, 1569-1571 (1993).
- [33] P. W. Shor, presented at *Quantum Computation 1994*, Villa Gualino, Turin, Italy, October 1994, unpublished.
- [34] D. Coppersmith, "An approximate Fourier transform useful in quantum factoring," IBM Research Report RC19642 (1994).
- [35] R. Cleve, "A note on computing Fourier transforms by quantum programs," University of British Columbia preprint (1994).
- [36] I. L. Chuang, R. Laflamme, P. W. Shor and W. H. Zurek, "Quantum computers, factoring and decoherence," *Science* **270**, 1633-1635 (1996).
- [37] In fact, we must be careful that the discrete Fourier transform yields sufficient resolution to extract the multiple of the inverse period from  $c/q$ . This is always possible provided the number of bits  $k$  in the first quantum register satisfies  $r^2 \leq q = 2^k$ .
- [38] W. H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing* (Cambridge University Press, Cambridge, 1988), pp. 390.
- [39] A. M. Odlyzko, "The future of integer factorization," AT&T Bell Laboratories preprint 1995.
- [40] R. Rivest, A. Shamir and L. Adleman, "A method of obtaining digital signatures and public-key cryptosystems," *Communications ACM* **21**, 120-126 (1978).
- [41] D. Atkins, M. Graff, A. K. Lenstra and P. C. Leyland, "The magic words are squeamish ossifrage," in *Advances in Cryptology - ASIACRYPT '94*, Eds. J. Pieprzyk and R. Safavi-Naini, Lecture Notes in Comp. Sci. 917 (Springer Verlag, Berlin, 1995), pp. 263-277.
- [42] O. Goldreich, *Foundations of Cryptography (Fragments of a book)*, available at <http://theory.lcs.mit.edu/~oded>.
- [43] See Ref. 38 pp. 228, where the DES algorithm is recommended as a pseudonumber generator for the simulation of physical processes.
- [44] A. M. Odlyzko, "Public key cryptography," *AT&T Tech. J.* **73**, 17-23 (1994).
- [45] B. Schneier, *Applied Cryptography*, (Wiley, New York, 1994).
- [46] D. Beckman, A. N. Chari, S. Devabhaktuni and J. Preskill, "Efficient networks for quantum factoring," *Phys. Rev. A* **54**, 1034-1063 (1996).
- [47] Since the period  $r$  is not known beforehand, we require  $N^2 \leq q = 2^k$  for the Fourier transform step to yield sufficient resolution [1, 10].
- [48] W. G. Unruh, "Maintaining coherence in quantum computers," *Phys. Rev. A* **51**, 992-997 (1995).
- [49] P. W. Shor, "Fault-tolerant quantum computation," in *Proceedings of the 37th Annual Symposium Foundations of Computer Science* (IEEE Comput. Soc. Press, Los Alamitos, CA, 1996), pp. 56-65.
- [50] A. M. Steane, "Error correcting codes in quantum theory," *Phys. Rev. Lett.* **77**, 793-797 (1996); "Multiple-particle interference and quantum error correction," *Proc. Roy. Soc. London* **452**, 2551-2576 (1997); "Simple quantum error-correcting codes," *Phys. Rev. A* **54**, 4741-4751 (1997).
- [51] A. R. Calderbank and P. W. Shor, "Good quantum error-correcting codes exist," *Phys. Rev. A* **54**, 1098-1105 (1996).
- [52] A. R. Calderbank, E. M. Rains, P. W. Shor and N. J. A. Sloane, "Quantum error correction and orthogonal geometry," *Phys. Rev. Lett.* **78**, 405-408 (1997).
- [53] R. Laflamme, C. Miquel, J. Pablo Paz and W. H. Zurek, "Perfect quantum error correcting code," *Phys. Rev. Lett.* **77**, 198-201 (1996).

- [54] E. Knill and R. Laflamme, "Theory of quantum error-correcting codes," *Phys. Rev. A* **55**, 900-911 (1997).
- [55] D. Gottesman, "Class of quantum error-correcting codes saturating the quantum Hamming bound," *Phys. Rev. A* **54**, 1862-1868 (1996).
- [56] There are several papers on fault tolerant computation: [49]; D. P. DiVincenzo and P. W. Shor, "Fault-Tolerant Error Correction with Efficient Quantum Codes," *Phys. Rev. Lett.* **77**, 3260-3263 (1996); M. B. Plenio, V. Vedral and P. L. Knight, "Conditional generation of error syndromes in fault-tolerant error correction," *Phys. Rev. A* **55**, 4593-4596 (1997); D. Aharonov and M. Ben-Or, "Fault Tolerant Quantum Computation with Constant Error," LANL preprint quant-ph/9611025; A. Steane, "Active stabilisation, quantum computation and quantum state synthesis," LANL preprint quant-ph/9611027; C. Zalka, "Threshold Estimate for Fault Tolerant Quantum Computation," LANL preprint quant-ph/9612028 and; D. Gottesman, "Theory of Fault-Tolerant Quantum Computation," *Phys. Rev. A* **57**, 127-137 (1998).
- [57] R. Jozsa, "Characterizing classes of functions computable by quantum parallelism," *Proc. R. Soc. Lond. A* **435**, 563-574 (1991).
- [58] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," *Proc. R. Soc. Lond. A* **439**, 553-558 (1992).
- [59] A. Chi-Chih Yao, "Quantum circuit complexity," in *Proceedings 34th Annual Symposium on Foundations of Computer Science* (IEEE New York, NY, 1993), pp. 352-361.
- [60] C. H. Bennett, E. Bernstein, G. Brassard and U. V. Vazirani, "Strengths and weaknesses of quantum computing," *Siam J. Comp.* **26**, 1510-1523 (1997).
- [61] S. Lloyd, "Universal quantum simulators," *Science* **273** 1073-1078 (1996).
- [62] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings, 28th Annual ACM Symposium on the Theory of Computing*, (ACM, New York, NY, 1996) pp. 212-219; L. K. Grover, "A fast quantum mechanical algorithm for estimating the median," LANL preprint quant-ph/9607024; M. Boyer, G. Brassard, P. Hoyer and A. Tapp, "Tight bounds on quantum searching," *Fort. Phys.* **46**, 493-505 (1998).
- [63] R. J. Hughes, presented at *Quantum Computation 1995*, Villa Gualino, Turin, Italy, June 1995, unpublished.
- [64] D. Loss and D. P. DiVincenzo, "Quantum computation with quantum dots," *Phys. Rev. A* **57**, 120-126 (1998).
- [65] M. B. Plenio and P. L. Knight, "Realistic lower bounds for the factorization time of large numbers on q quantum computer," *Phys. Rev. A* **53**, 2986-2990 (1996).
- [66] R. J. Hughes, D. F. V. James, E. H. Knill, R. Laflamme and A. G. Petschek, "Decoherence bounds on quantum computation with trapped ions," *Phys. Rev. Lett.* **77**, 3240-3243 (1996).
- [67] A. Steane, "The ion trap quantum information processor," *Appl. Phys. B* **64**, 623-642 (1997).
- [68] M. B. Plenio and P. L. Knight, "Decoherence limits to quantum computation using trapped ions," *Proc. Roy. Soc. Lond. A* **453**, 2017-2041 (1997).
- [69] G. H. Hardy and E. M. Wright, *An introduction to the theory of numbers* (Oxford, Clarendon Press, 1979).